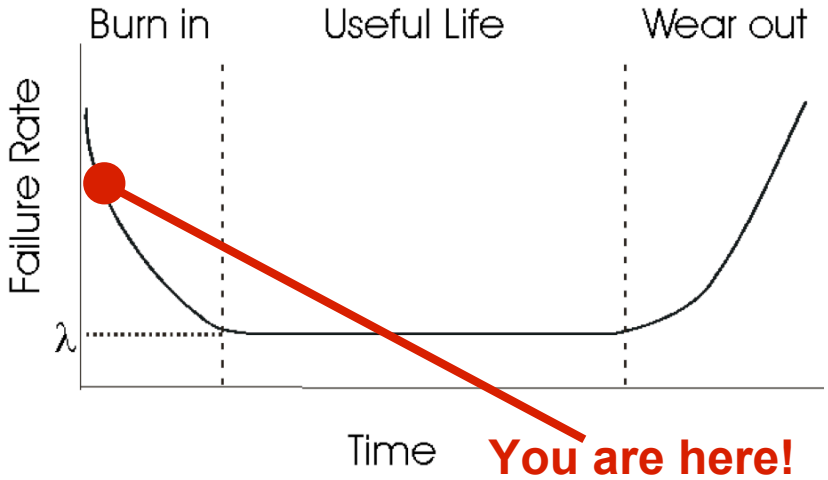
Big Science Meets the Bathtub Curve

John T. Daly

Roadrunner Open Science Presentations
Los Alamos, April 23, 2008

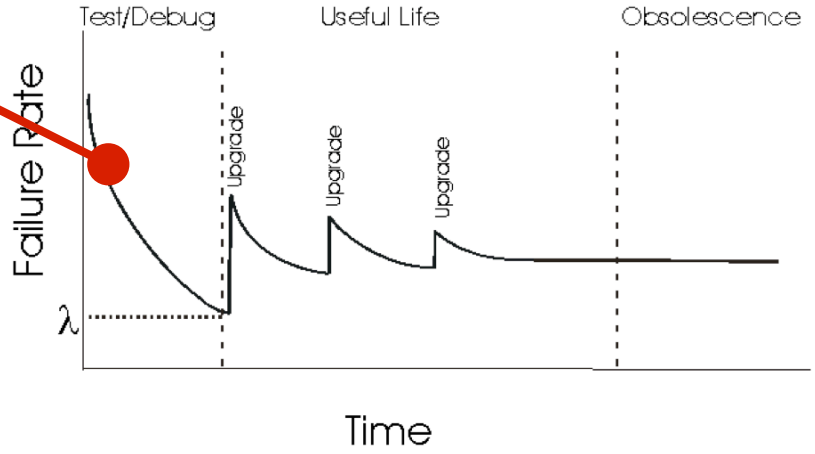
jtd@lanl.gov

“Early Adopters” experience the highest failure rates in the operational lifetime of a system*



Hardware reliability improves over time and remains fairly constant until end of life when components begin to wear out

Sofware reliability improves piecewise over time but faces periodic downturns because of the bugs and increasing complexity associated with upgrades



* http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/

“Early Adopters” must find ways to make calculations succeed in a failure rich environment

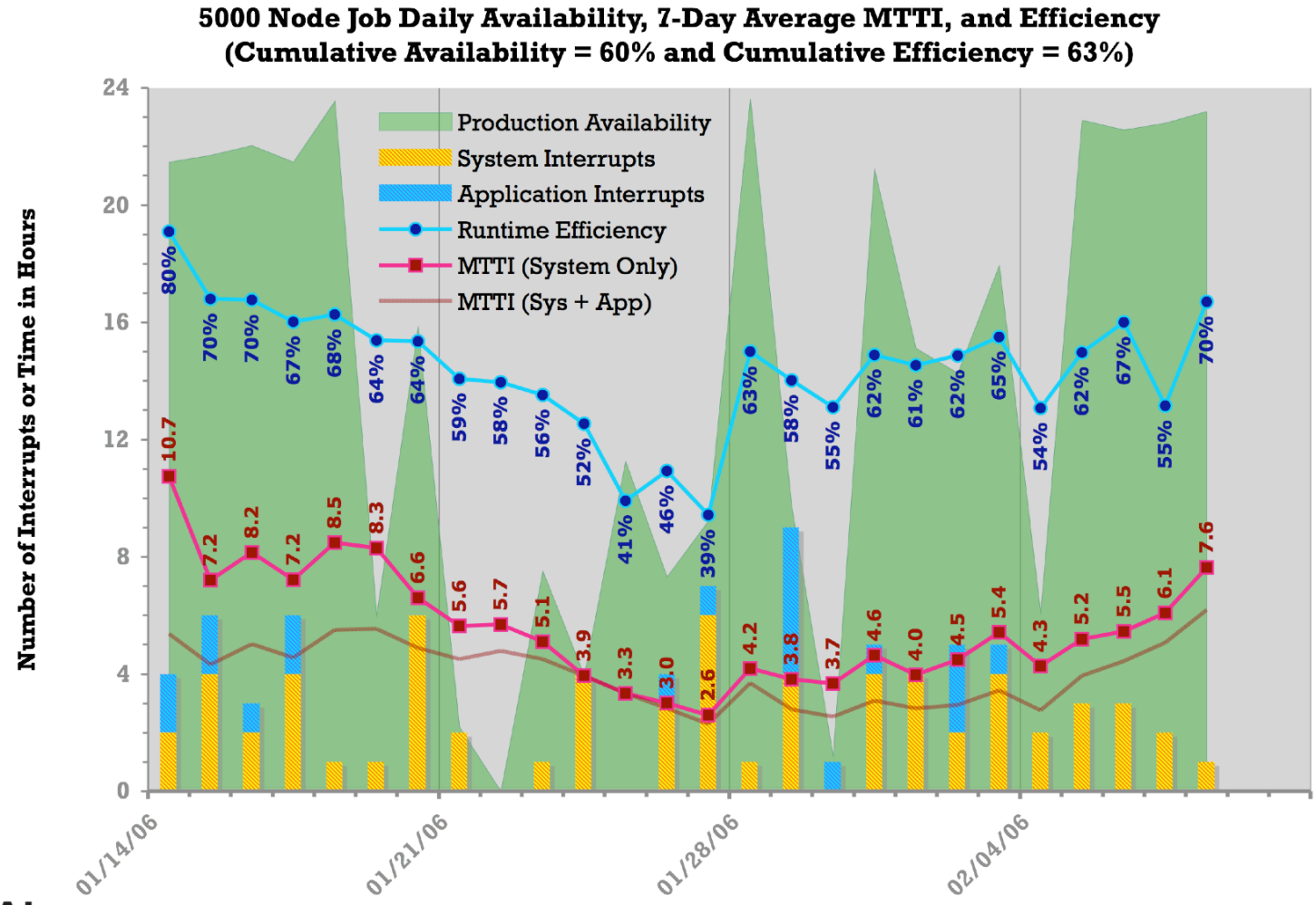
Thomas Edison was a firm believer in the importance of failure. Failure was what led him to success. Once, when he was working on developing a better battery, a discouraged assistant came up to him and suggested that Mr. Edison must be ready to quit after having performed 50,000 tests without success. Despite the assistant’s ideas of quitting Edison felt he made progress. “At least we know 50,000 things that won't work!” In the end he developed a nickel-iron alkaline battery that is still used today-more than 90 years later!

from http://www.bkfk.com/inventor/failed.asp

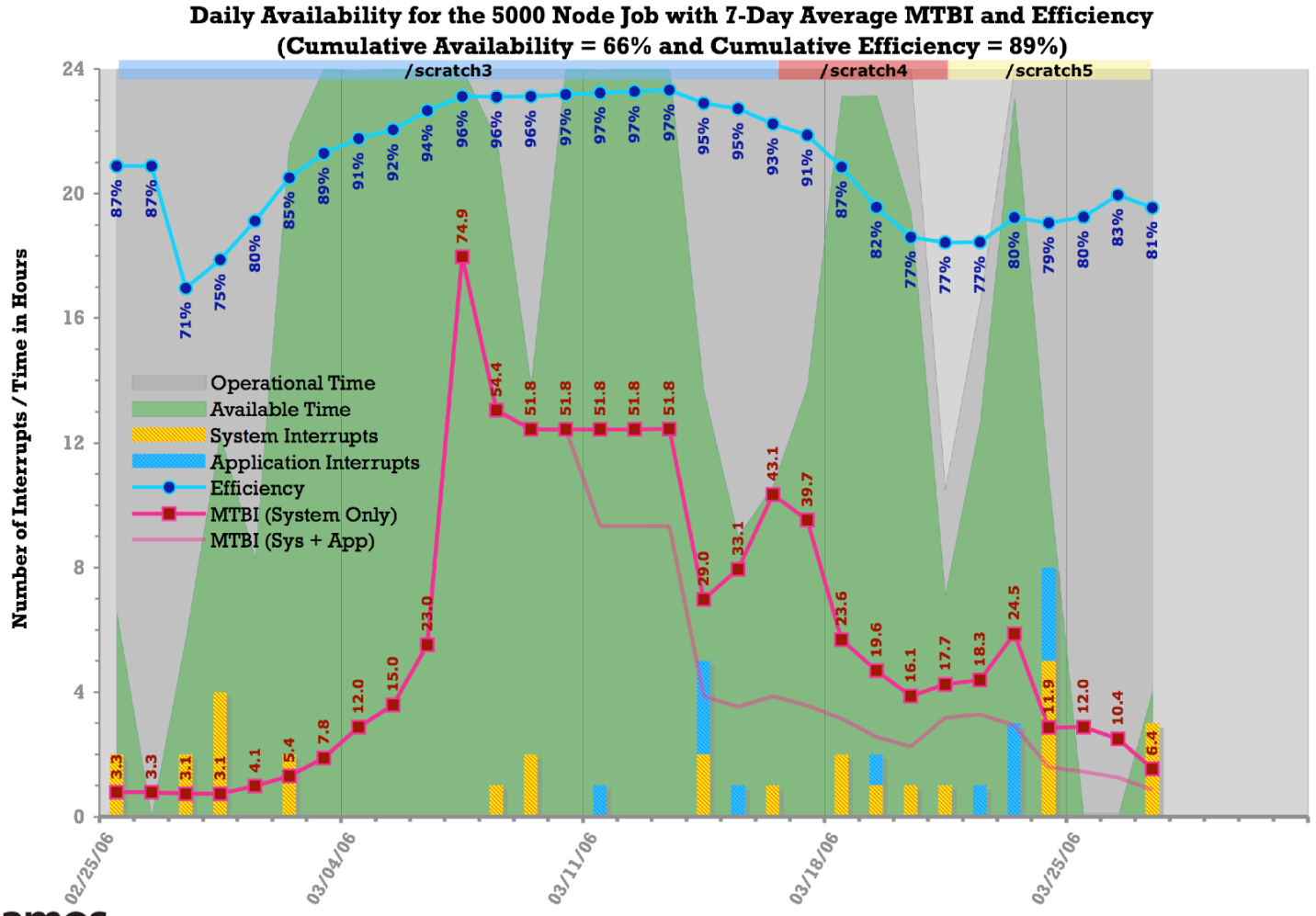
Big Science Meets the Bathtub Curve -- Part 1

- **Motivation: Data Gathered by an "Early Adopter"**
- **Characterizing the Throughput of an Application**
  - Performance and Scaling
  - Efficiency and Utilization
- **Quantify the Impact of Reliability on Throughput**
  - Optimum Checkpointing Interval
  - Runtime Efficiency
- **Strategies to Improve Application Throughput**
  - Maximize Ratio of MTTI to Dump Time
  - Minimize Restart Overhead
- **Results Using Application Monitoring**

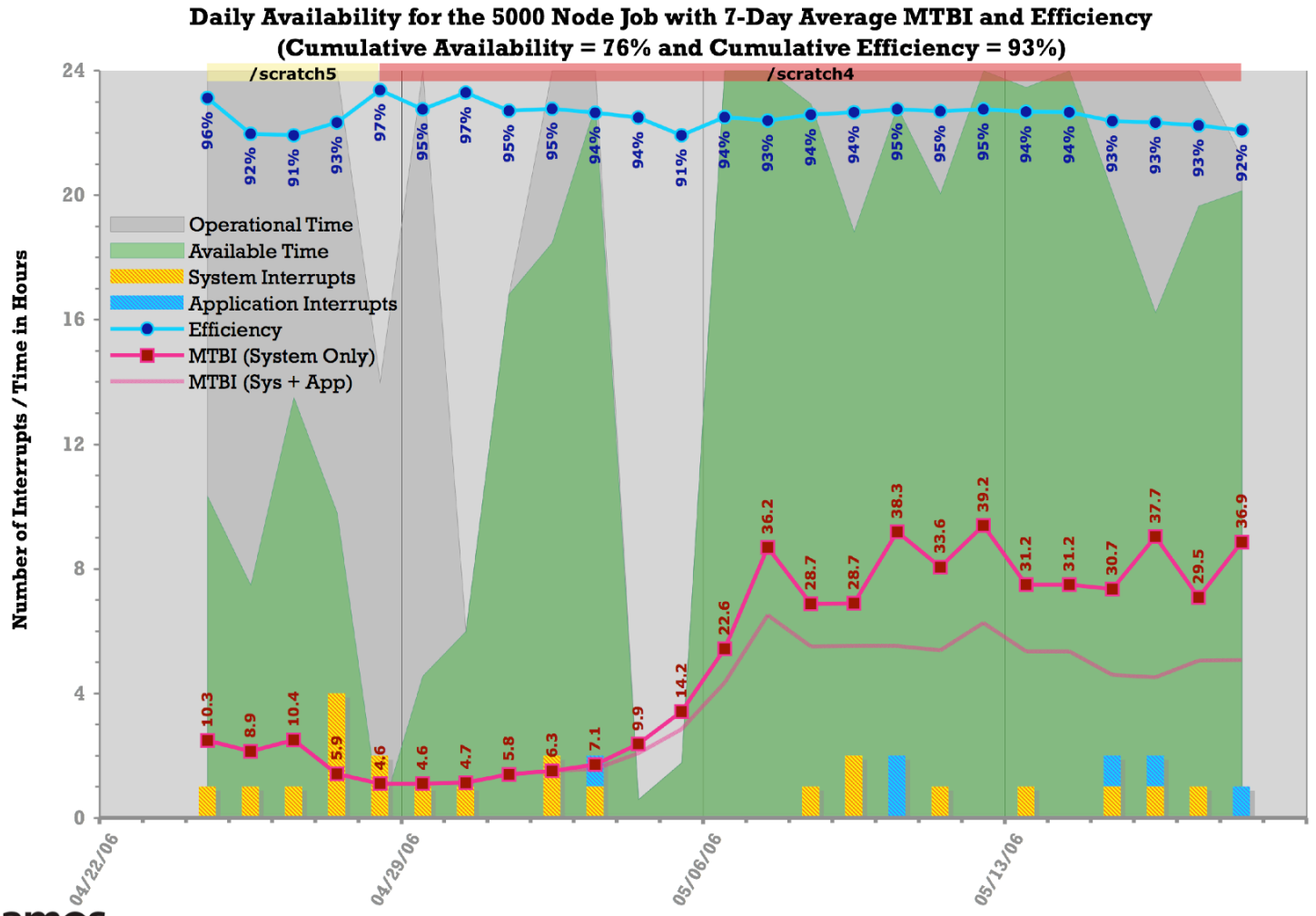
“Early Adopter” experience of Red Storm: Reliability was dicey in the beginning



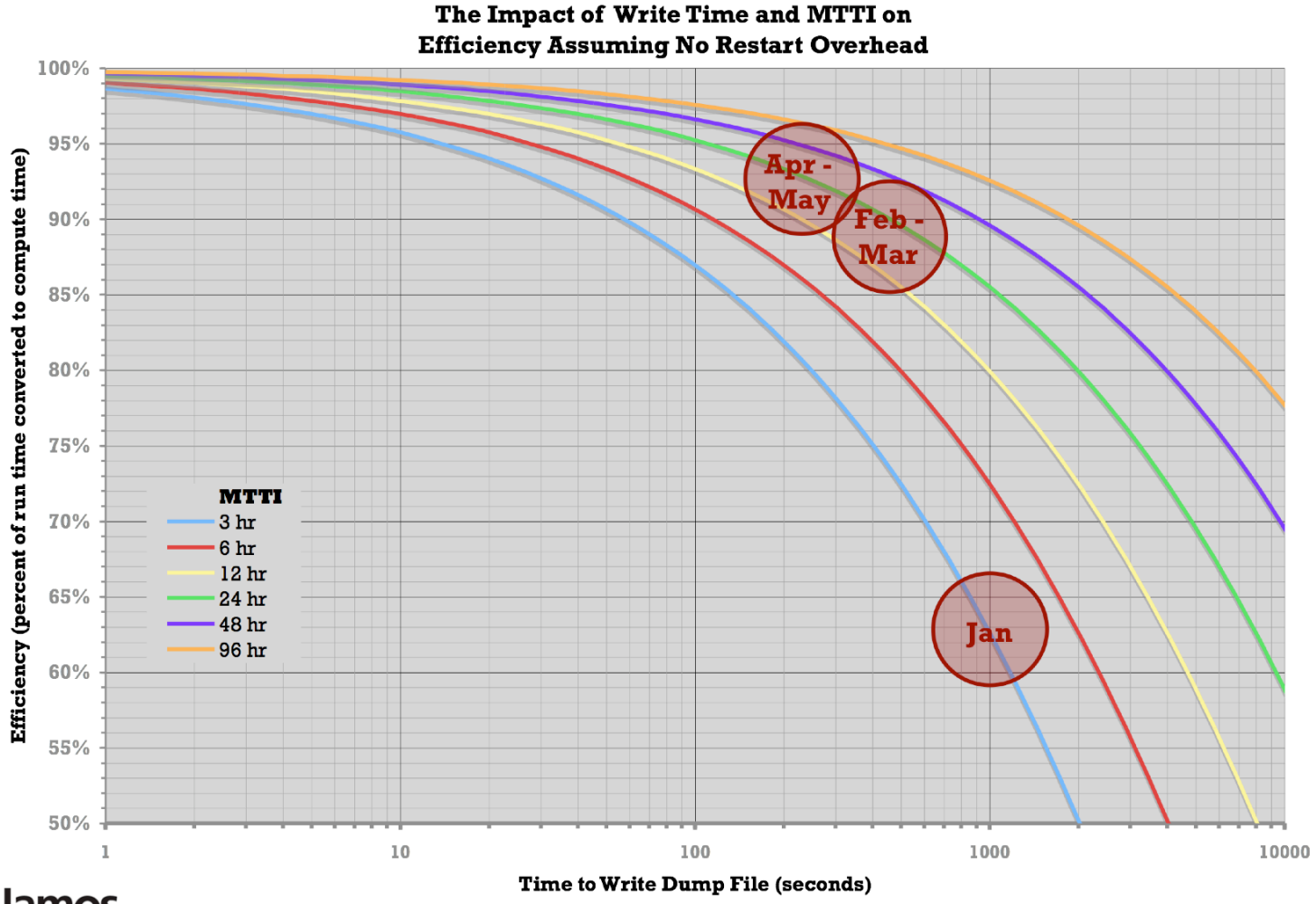
“Early Adopter” experience of Red Storm: Within a month or two reliability improved dramatically



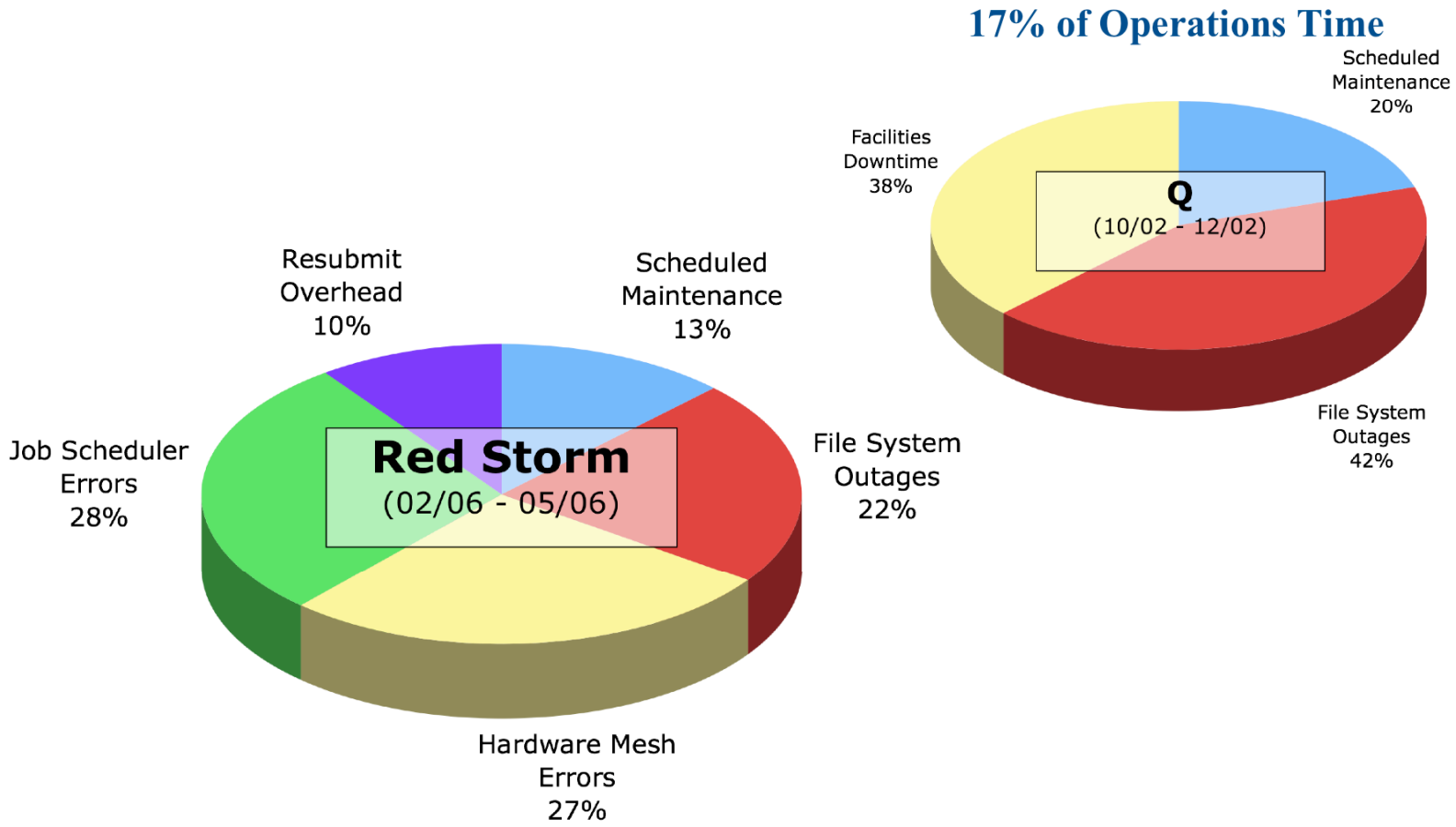
“Early Adopter” experience of Red Storm: Even after four months there were still periods of low availability



Improving write rates and MTTI dramatically increases computational efficiency over time



The fraction and causes of downtime on Red Storm were consistent with observations of the Q machine



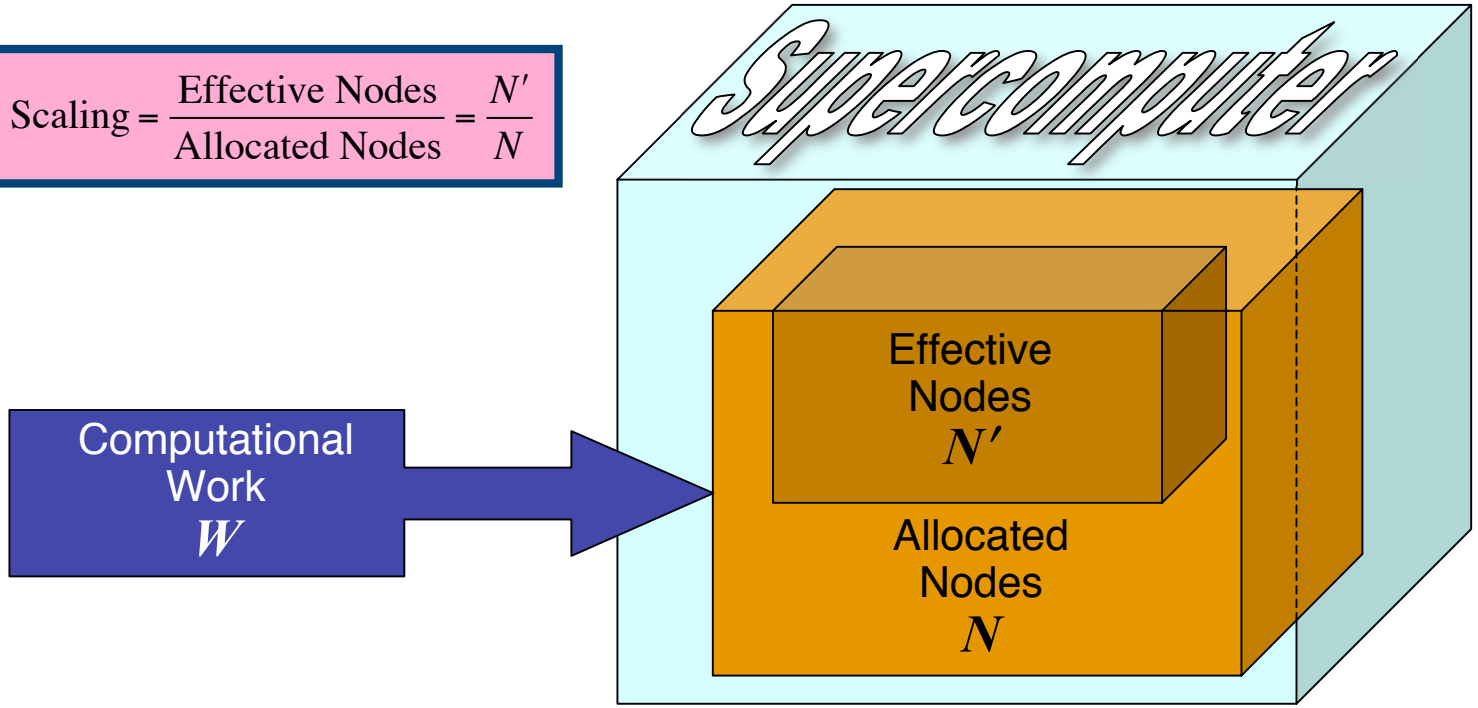
Big Science Meets the Bathtub Curve -- Part 2

- Motivation: Data Gathered by an "Early Adopter"
- **Characterizing the Throughput of an Application**
  - Performance and Scaling
  - Efficiency and Utilization
- Quantify the Impact of Reliability on Throughput
  - Optimum Checkpointing Interval
  - Runtime Efficiency
- Strategies to Improve Application Throughput
  - Maximize Ratio of MTTI to Dump Time
  - Minimize Restart Overhead
- Results Using Application Monitoring

Operations rate: a traditional benchmark for measuring application throughput

$$\text{Node Performance} = \frac{\text{Computational Work}}{\text{Solve Time} \cdot \text{Effective Nodes}} = \frac{W}{t_s \cdot N'}$$

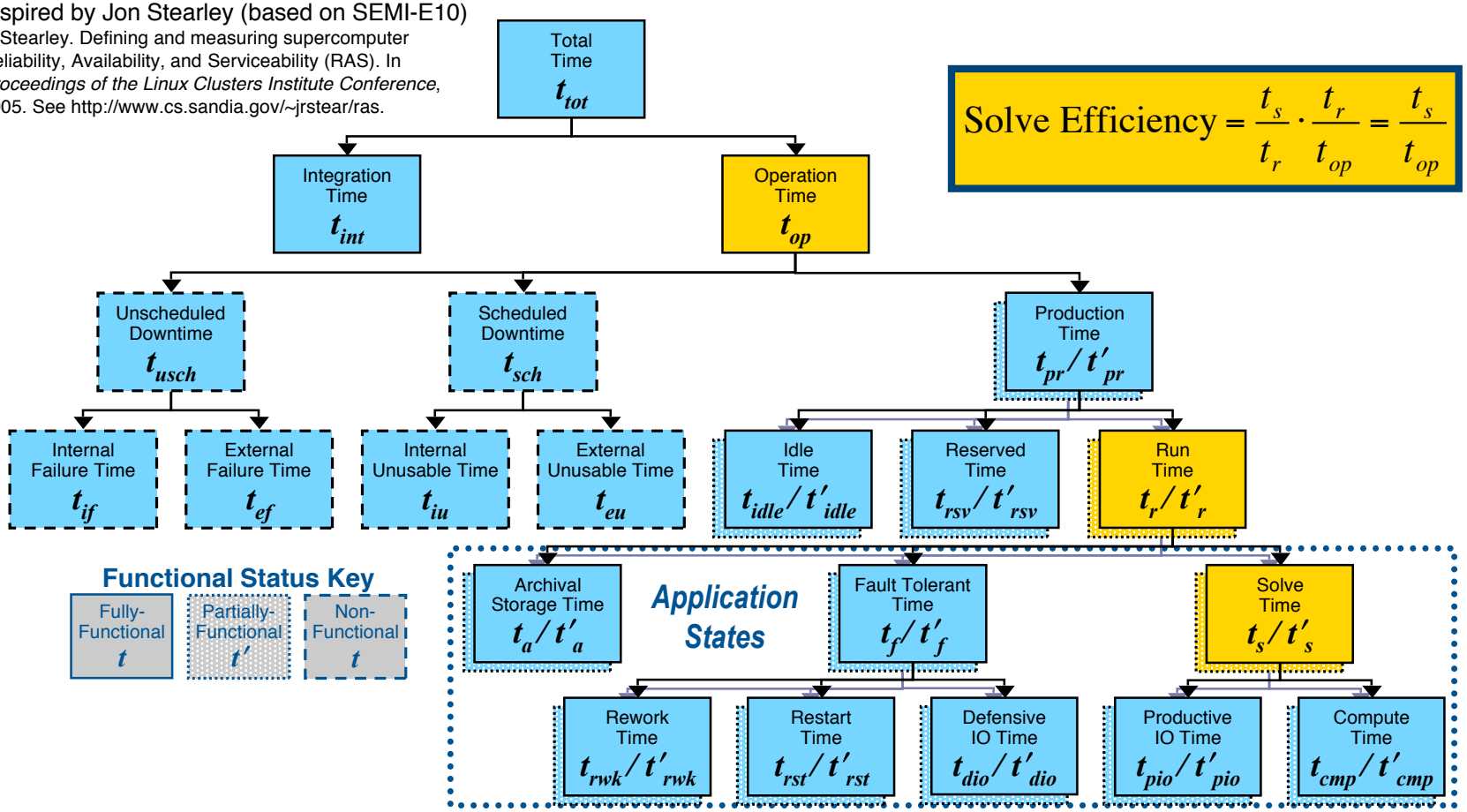
$$\text{Parallel Scaling} = \frac{\text{Effective Nodes}}{\text{Allocated Nodes}} = \frac{N'}{N}$$



$$\text{Operations Rate} = \text{Performance} \cdot \text{Scaling}$$

However, operations rate only considers time spent in a fraction of the possible system states

Inspired by Jon Stearley (based on SEMI-E10)
J. Stearley. Defining and measuring supercomputer Reliability, Availability, and Serviceability (RAS). In *Proceedings of the Linux Clusters Institute Conference*, 2005. See http://www.cs.sandia.gov/~jrstear/ras.



$$\text{Solve Efficiency} = \frac{t_s}{t_r} \cdot \frac{t_r}{t_{op}} = \frac{t_s}{t_{op}}$$

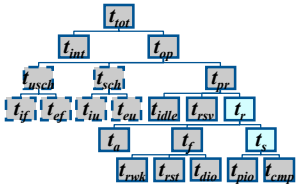
* Proposed in collaboration with S. Michalak and L. Davey

Application throughput (compute per time): Depends on reliability as well as performance

- **The amount of *computational work* completed over a period of time during which a system is operational, (i.e. *operations time*), though not necessarily available, is the *application throughput***

$$\text{Operations Time} = \frac{\text{Solve Time}}{\text{Efficiency} \cdot \text{Utilization}}$$

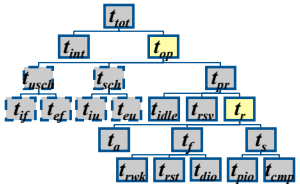
- **The fraction of run time during which the application is making progress towards a solution is its *run time efficiency***



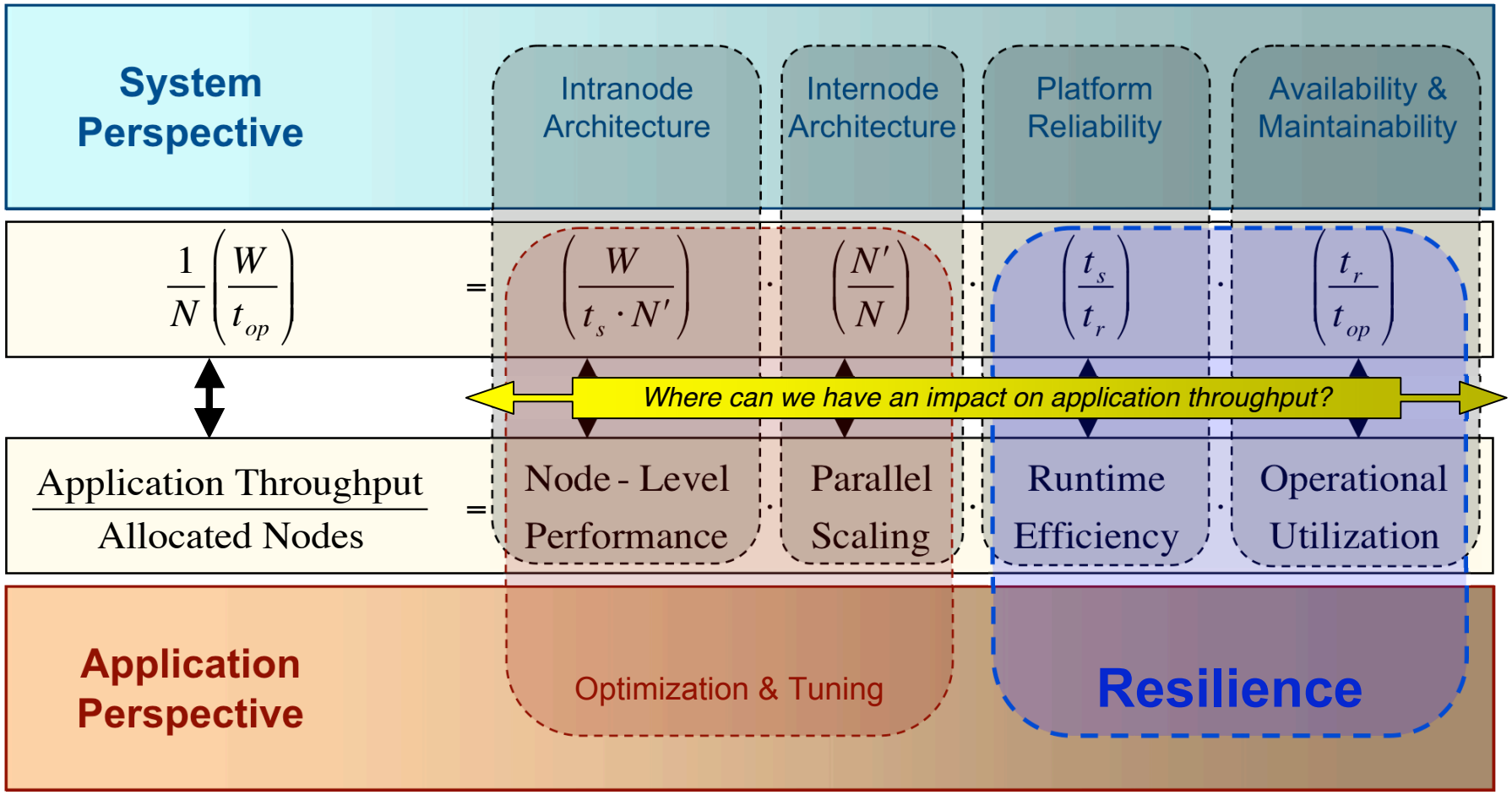
$$\text{Run Time Efficiency} = \frac{\text{Solve Time}}{\text{Run Time}} = \frac{t_s}{t_r}$$

- **The fraction of operational time during which the application is running is its *operational utilization***

$$\text{Operational Utilization} = \frac{\text{Run Time}}{\text{Operations Time}} = \frac{t_r}{t_{op}}$$



Throughput can be tuned by performance and reliability from both the system's and application's perspectives

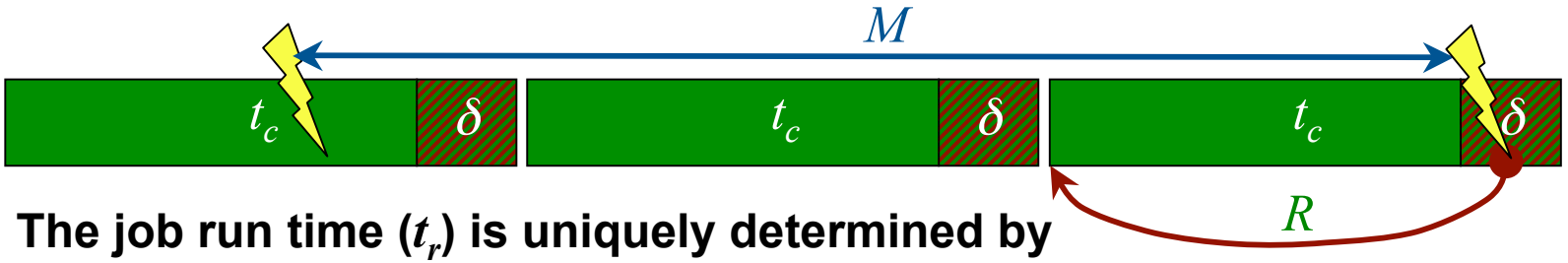


* J.T. Daly and K. A. Winkler, *Los Alamos Technical Report* (LA-UR-07-2626)

Big Science Meets the Bathtub Curve -- Part 3

- Motivation: Data Gathered by an "Early Adopter"
- Characterizing the Throughput of an Application
  - Performance and Scaling
  - Efficiency and Utilization
- **Quantify the Impact of Reliability on Throughput**
  - Optimum Checkpointing Interval
  - Runtime Efficiency
- Strategies to Improve Application Throughput
  - Maximize Ratio of MTTI to Dump Time
  - Minimize Restart Overhead
- Results Using Application Monitoring

Solve Time vs. Run Time*: Quantifying the effectiveness of recovery oriented methods for fault-tolerance



- **The job run time (t_r) is uniquely determined by**
  - Solve time (t_s) -- time required to complete an uninterrupted job
  - Application MTTFE (M) -- expected time before the next interrupt
  - Dump Time (δ) -- overhead required to save a known good state
  - Checkpoint Interval (t_c) -- time elapsed between saving states
  - Restart Overhead (R) -- time to restart from a previous saved state
- **Assuming that failure interarrival times are distributed exponentially, the application run time will be**

$$t_r = Me^{\frac{R}{M}}\left(e^{\frac{t_c}{M}} - 1\right)\frac{t_s}{t_c - \delta} \quad \text{for} \quad \delta << t_s$$

* J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps", Future Generation Computer Systems 22 (2006) 300-312

Optimum checkpoint interval and runtime efficiency can be defined by a single non-dimensional parameters

- **Done by solving the minimization problem**

$$e^{-\lambda} + \lambda = \tfrac{1}{2}\Delta^2 + 1 \quad \text{where} \quad \lambda = \frac{t_c}{M},\ \Delta = \sqrt{\frac{2\delta}{M}}$$

- **This can be solved exactly using Lambert's W-function**

$$\lambda = \tfrac{1}{2}\Delta^2 + 1 + W\left(-e^{-\frac{1}{2}\Delta^2 - 1}\right)$$

$$t_c = \sqrt{2\delta M}$$

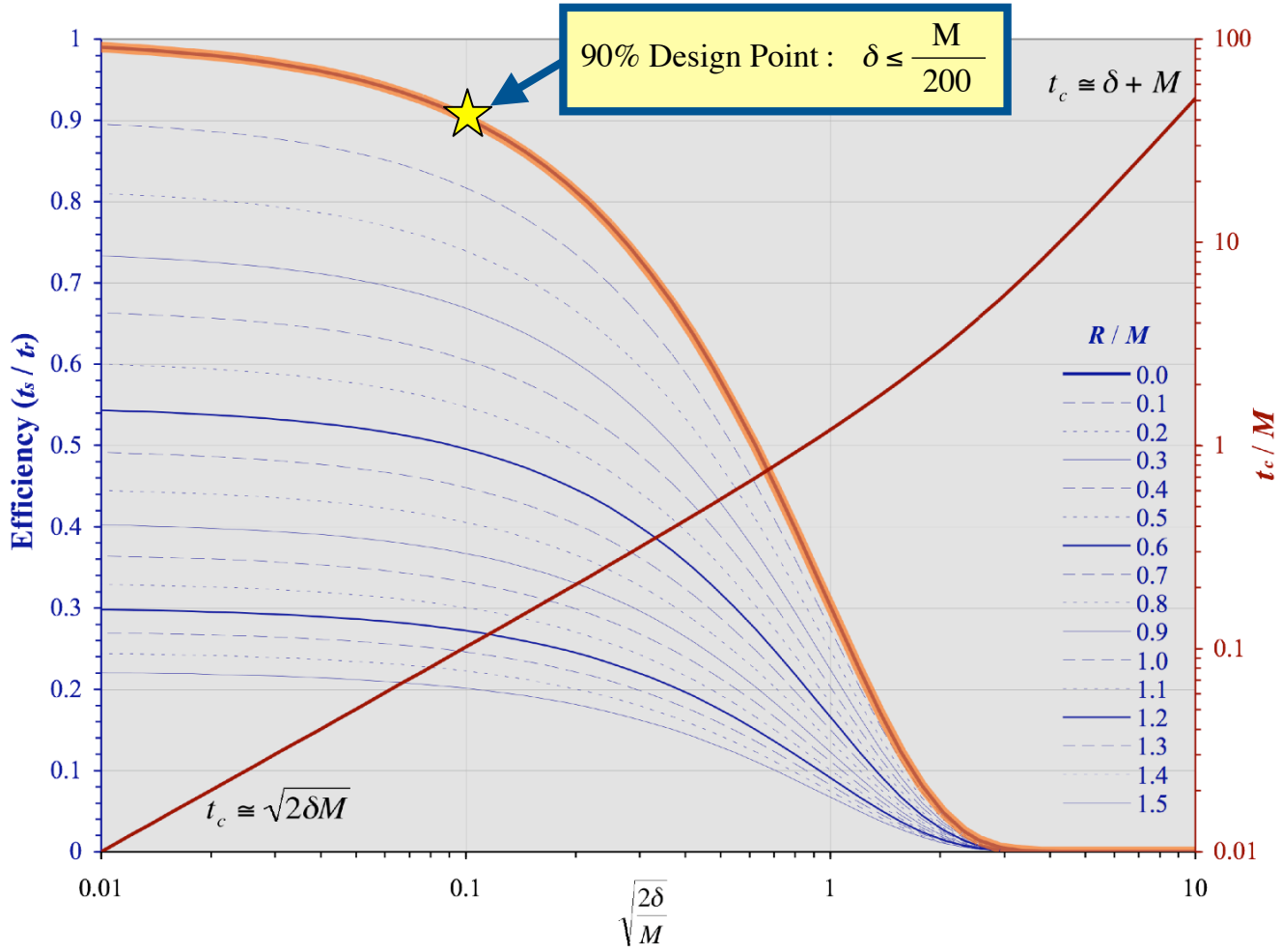
- **Or, using a perturbation solution and an asymptotic solution**

$$\lambda = \begin{cases} \Delta + \frac{1}{6}\Delta^2 + \frac{1}{36}\Delta^3 + \frac{1}{270}\Delta^4 + O\left(\Delta^5\right) & \text{for} \quad \Delta \le 2.55 \\ \frac{1}{2}\Delta^2 + 1 & \text{for} \quad \Delta > 2.55 \end{cases}$$

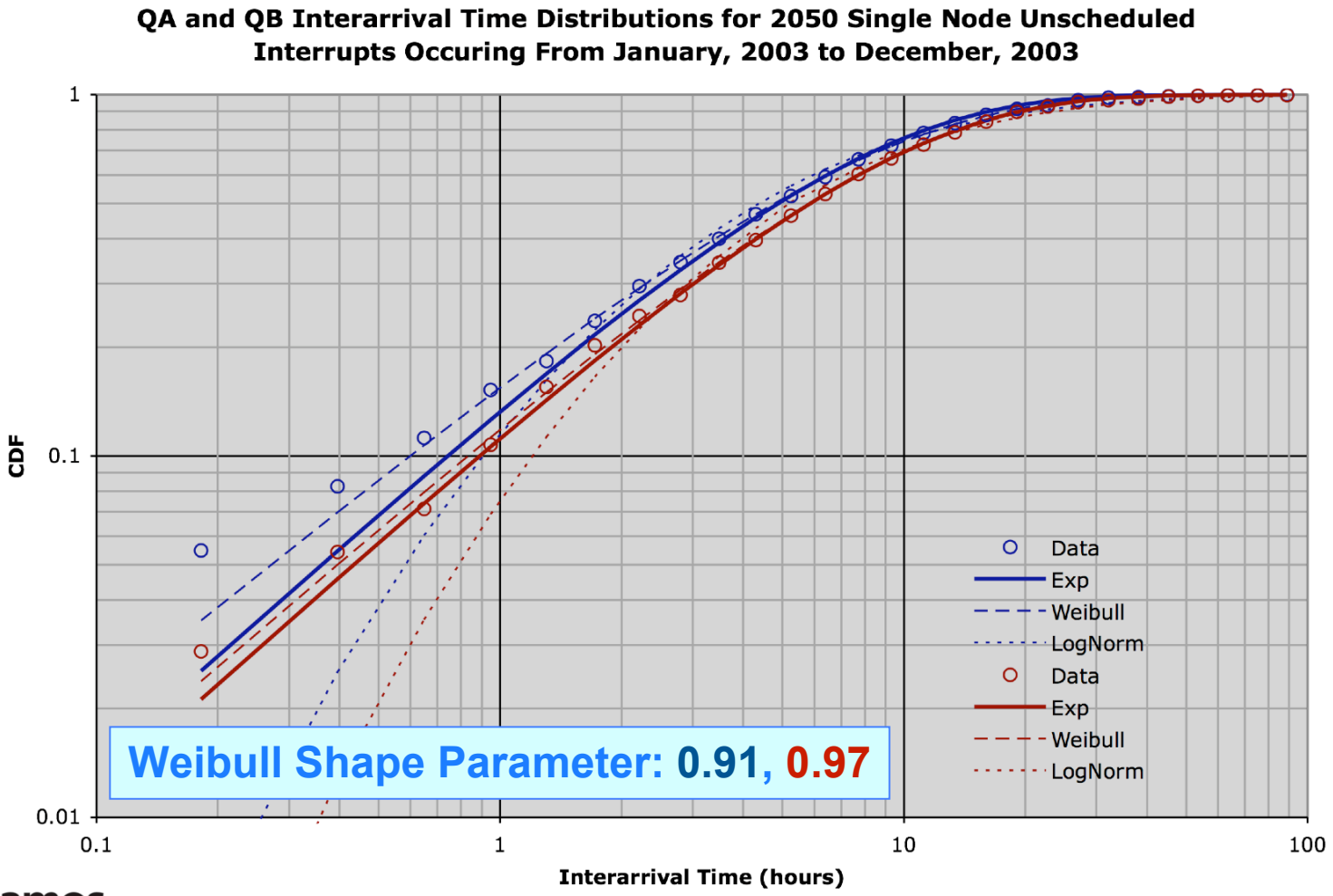
- **Now runtime efficiency can be expressed in terms of checkpoint interval and dump time**

$$\text{Efficiency} = e^{-\frac{R}{M}}\left(\frac{\lambda - \frac{1}{2}\Delta^2}{e^{\lambda} - 1}\right)$$

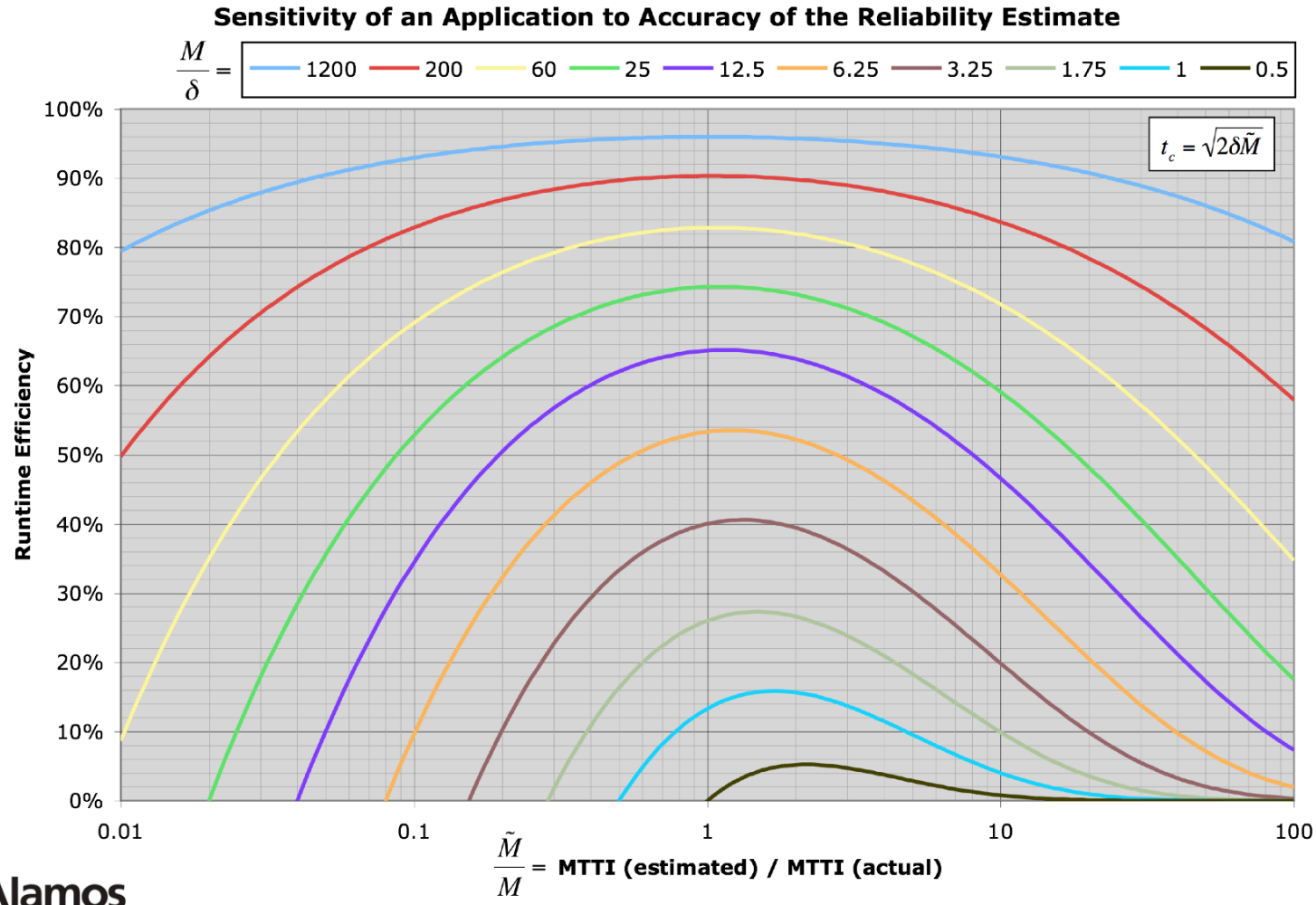
Efficiency of checkpoint fault tolerance is determined by reliability, dump time, and restart overhead only



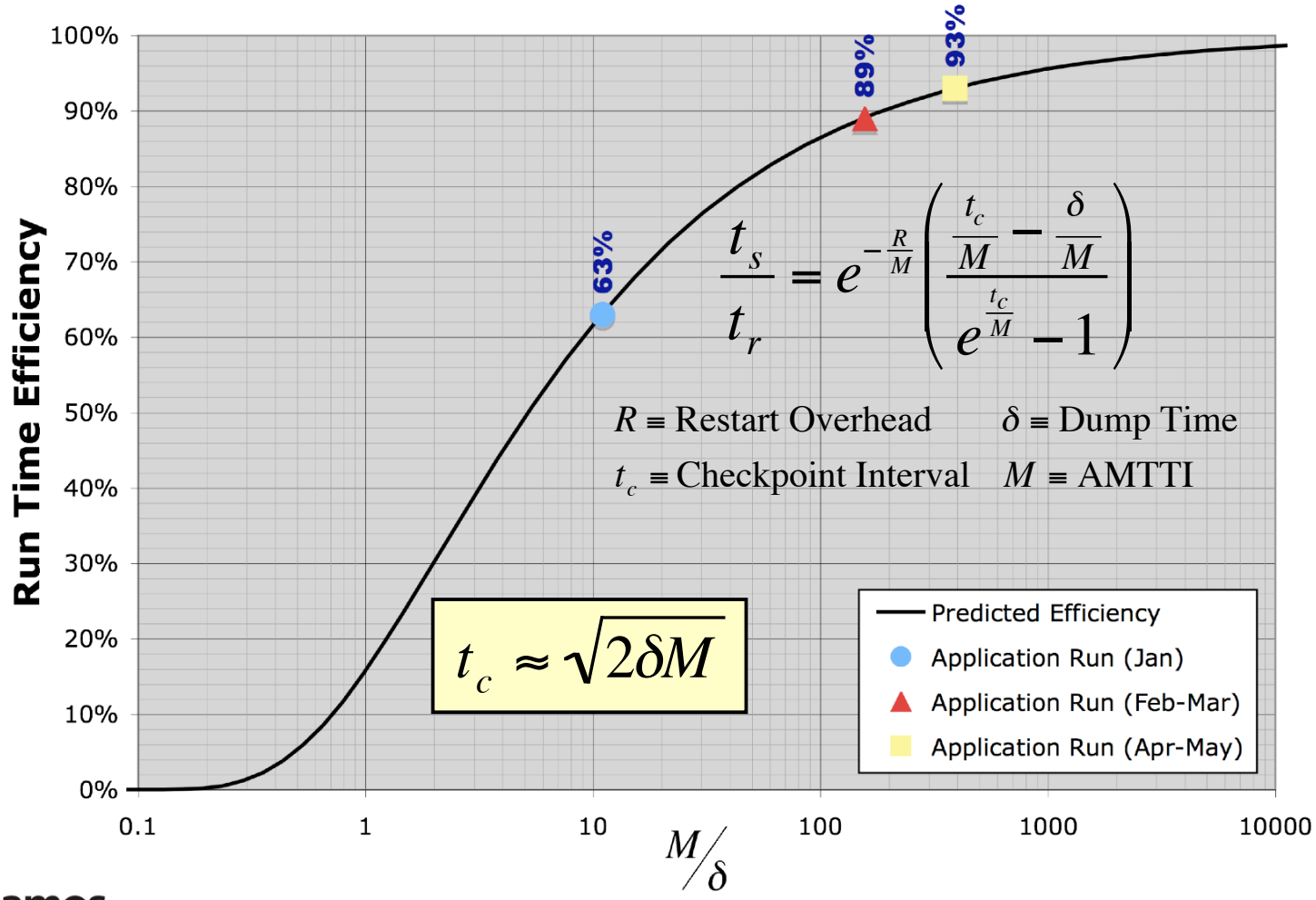
LANL failure data can be used to demonstrates nearly exponential behavior in the case of many systems



Additionally, the efficiency is not overly sensitive to reliability estimates regardless of failure distribution



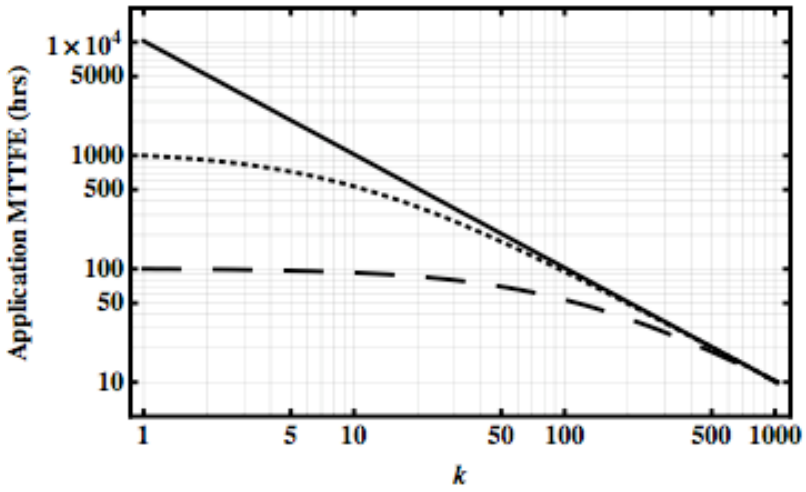
Sanity Check: Runtime efficiency model accurately reflects "early adopter" data from Red Storm



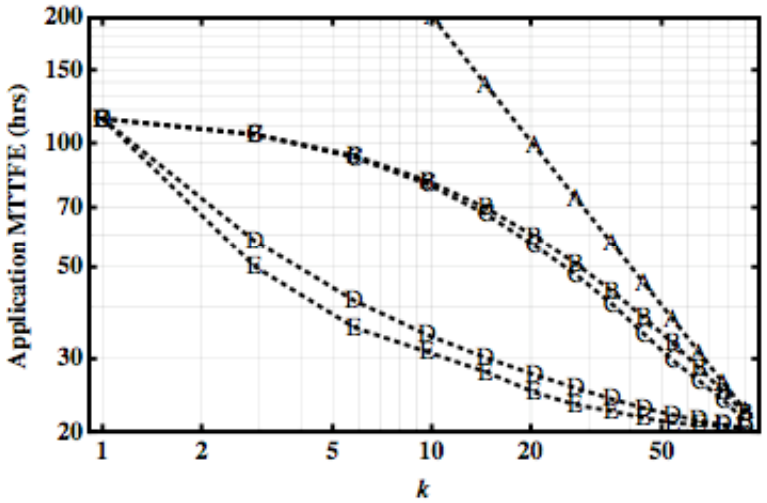
Big Science Meets the Bathtub Curve -- Part 4

- Motivation: Data Gathered by an "Early Adopter"
- Characterizing the Throughput of an Application
  - Performance and Scaling
  - Efficiency and Utilization
- Quantify the Impact of Reliability on Throughput
  - Optimum Checkpointing Interval
  - Runtime Efficiency
- **Strategies to Improve Application Throughput**
  - Maximize Ratio of MTTI to Dump Time
  - Minimize Restart Overhead
- Results Using Application Monitoring

Application mean time to fatal error* is a better metric than system MTBF for quantifying the user's experience



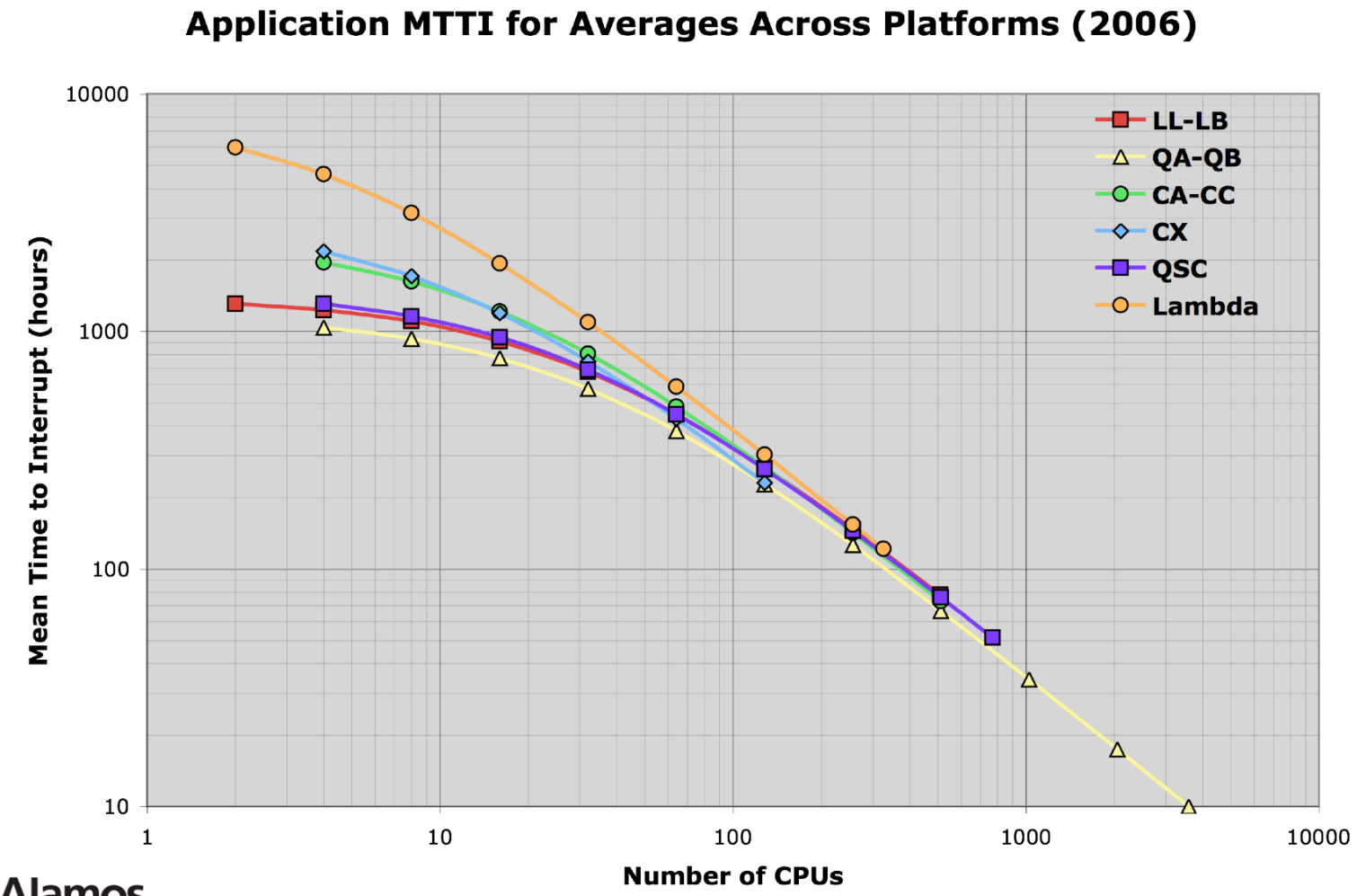
First order approximation of application mean time to fatal error demonstrates super-linear per processor reliability scaling



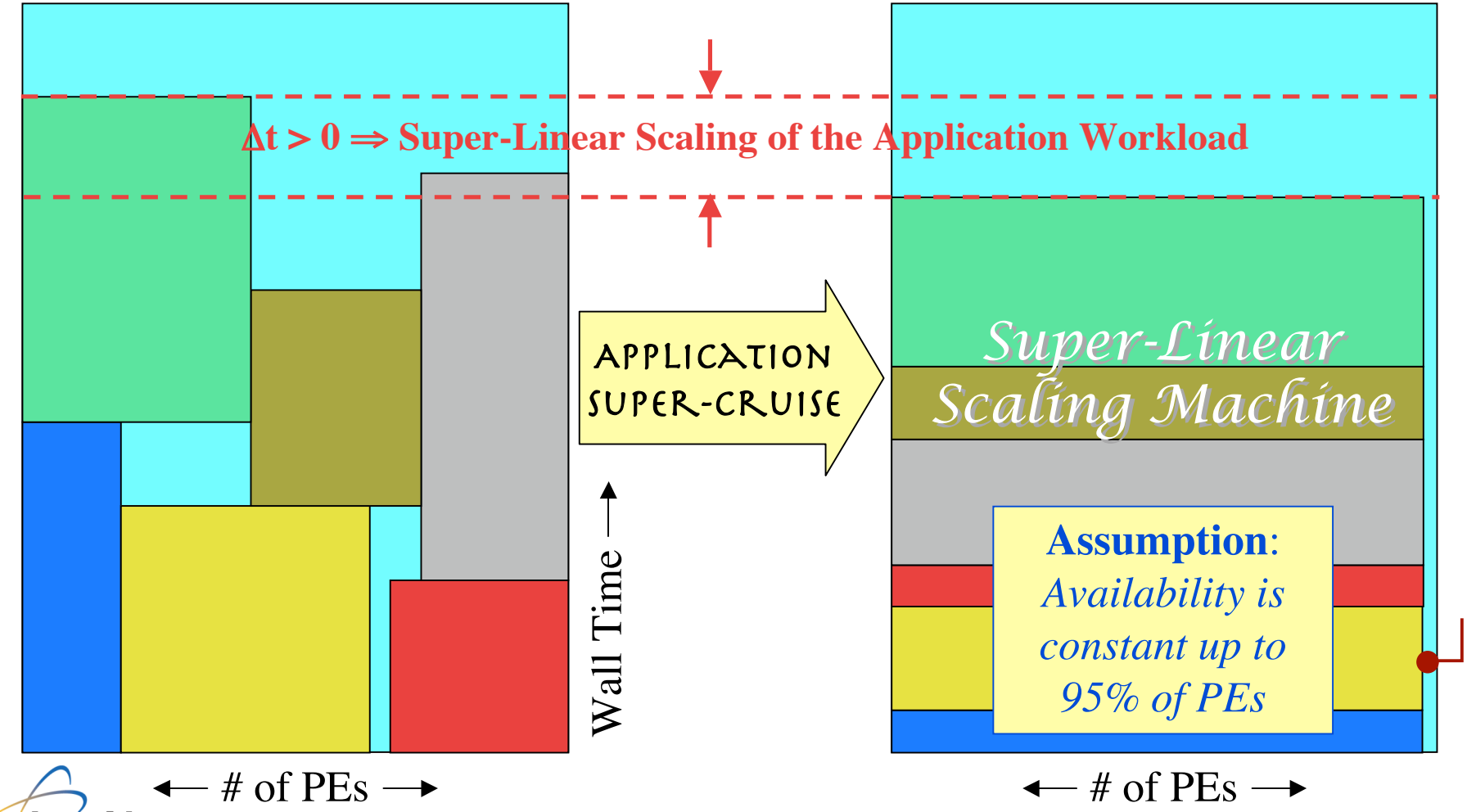
A -- Inverse Proportionality
B -- First Order Approximation
C -- Exact (Contiguous Nodes)
D -- Exact (Random Nodes)
E -- Exact (Worst Case Nodes)
k -- number of processors

* Daly, Pritchett-Sheats, and Michalak, "Application MTTFE vs. platform MTBF: a fresh perspective on system reliability and application throughput for computations at scale", proceedings of Workshop on Resilience in HPC, CCGrid (2008) *forthcoming*

Analysis of 140,000 interrupt events confirms that per PE behavior is extremely consistent across platforms



Maximize the ratio of MTTFE to dump time: as reliability plateaus the workload can even scale *super-linearly*

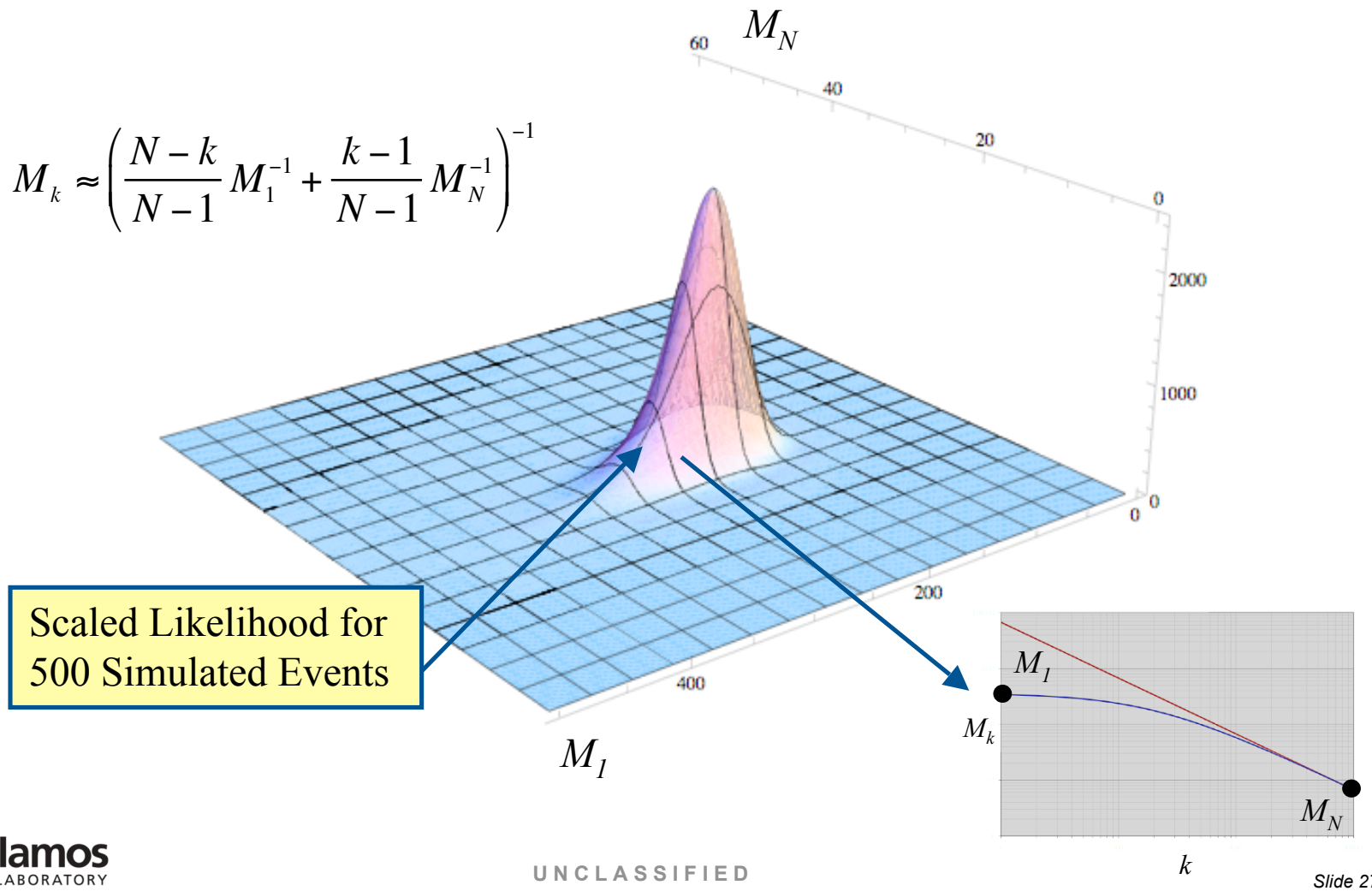


Minimize restart overhead between jobs: *Job Suite* is a portable and robust toolkit for application monitoring

- **The goal of application monitoring is to minimize the time required to get an application running again after a fatal error**
- ***Job suite* is a python based toolset developed to maximize utilization of DOE capability class compute platforms**
  - Interacts with applications through an arbitrary system job scheduler (currently supports LSF, Moab, PBS Pro, and LCRM) or schedules its own jobs in a Posix environment
  - Monitors application progress and reduces user workload by checking for updates to stdout or a specified output file
  - Demonstrated on over 40 million CPU hours of production computations on Red Storm, Purple, and BG/L
  - Includes tools to *view*, *list*, *kill*, *modify*, and *restart* jobs
- **Data collected from *job suite* can also be used to estimate reliability**

Example: Simulated job *runtime*, *processor count*, and *exit status* data is used to estimate the AMTTFE curve

$$M_k \approx \left(\frac{N-k}{N-1} M_1^{-1} + \frac{k-1}{N-1} M_N^{-1} \right)^{-1}$$



The *job suite* tool is highly configurable to support a broad range of applications and system

- **List of available options (required and important)…**

```
-v          -- print the version number and
    return
-B|-BB      -- (LSF only) a job receiving a
    bkill will only be restarted if the job
    idle time limit (see -t) has been exceeded
    (-B turns this option 'on' and -BB turns it
    'off') [default: 'off']
-F|-FF      -- skip the status evaluation step
    at job termination and force a restart (see
    -k for exception) regardless of the status
    (-F turns this option 'on' and -FF turns it
    'off') [default: 'off']
-W|-WW      -- (POE only) watch progress of the
    monitor file from the job's head node to
    avoid buffering by the application launcher
    (-W turns this option 'on' and -WW turns it
    'off') [default: 'on']
-E|-EE      -- send e-mail notification when a
    job is completed
-H|-HH      -- hold jobs still in the scheduler
    queue, instead of killing them, when
    cleaning up after an exit caused by errors
    in the current job or session
-w <arg>    -- length of time that the job
    submission script will continue attempting
    to submit the job (DD:HH:MM or minutes)
-t <arg>    -- length of time that the job
    submission script will wait on an idle job
    before killing it (DD:HH:MM or minutes)
-p <file>   -- full path name of the primary
    run script to be submitted on the first
    execution of the job
-pa <list>  -- arguments to the primary run
    script
-s <file>   -- full path name of the secondary
    run script to be submitted on all
    subsequent executions of the job
```

The *job suite* tool is highly configurable to support a broad range of applications and system

- **And more options (required and important)…**

```
-sa <list>  -- arguments to the secondary run
    script
-a <list>   -- list of command line arguments
    to pass to the job scheduler
-b <arg>    -- limits the number of consecutive
    jobs that can either hang or run for less
    than the idle time limit (see -t) before
    the session will bail out (set 0 to ignore)
    [default: 0]
-d <dir>    -- directory where the job monitor
    file containing the unique job id as part
    of its name is being written
-o <dir>    -- output directory to monitor for
    changing numbers of files
-J <arg>    -- specify the job scheduler to be
    used
-A <arg>    -- specify the application launcher
    to be used
-h <arg>    -- length of time to hold between
    successive jobs (DD:HH:MM or minutes)
    [default: 0, min: 0]
-T <arg>    -- length of time to sleep before
    beginning to submit any jobs (DD:HH:MM or
    minutes) [default: 0, min: 0]
-M <arg>    -- monitoring interval for checking
    whether the job monitor file size has
    increased (minutes) [default: 5, min: 0.5,
    max: 60]
-P <arg>    -- (no POSIX) polling interval for
    determining if the monitored job is still
    running (seconds) [default: 60, min: 1,
    max: 600]
-n <arg>    -- put a limit on the maximum
    number of jobs that the submit script will
    launch before exiting [default: 100, min:
    1]
-R <arg>    -- length of time required to
    elapse before the internal queue can be
    refilled (DD:HH:MM or minutes) [default:
    60, min: 0]
```

The *job suite* tool is highly configurable to support a broad range of applications and system

- **And even more options (required and important)…**

```
-l <arg>    -- number of internal queue slots
    which determines how many jobs will be
    submitted at one time [default: 2, min: 1,
    max: 20]

-j <list>   -- list of job ids that job_submit
    will attempt to monitor before submitting
    any new run scripts (see -p) to the queue

-r <arg>    -- restart a session with an
    existing restart file (<id #> on local host
    or <id #>.<host>)

-m <arg>    -- modify arguments for a currently
    active session (<id #> on local host or <id
    #>.<host>)

-c <arg>    -- prepends the crontab shell
    command with the supplied string

-D <arg>    -- set the debug level: 0 - none, 1
    - low, 2 - high [default: 0]

-V <arg>    -- set the verbosity level: 0 - no
    output, 1 - write the log file only, 2 -
    write to stdout only, 3 - write to both
    stdout and the log file [default: 3]

-C <arg>    -- set the frequency at which a
    cron job will check the current job submit
    session and attempt to restart it when no
    longer active: 0 - no cron job, 1 - every 6
    hrs, 2 - every 3 hrs, 3 - every hour, 4 -
    every minute [default: 0]

-S <arg>    -- set a limit on the allowed
    number of active sessions per host
    [default: 20, min: 1, max: 100]

-L <dir>    -- specify the directory to which
    the log file will be written [default:
    $HOME/.job_submit]

-k <list>   -- provide a list of script status
    return values that will cause job_submit to
    exit (overrides the -F option)

-q <list>   -- provide a list of job scheduler
    queue names to be used when running
    "restricted" visibility commands [default:
    all queues available to $USER]
```

Big Science Meets the Bathtub Curve -- Part 5

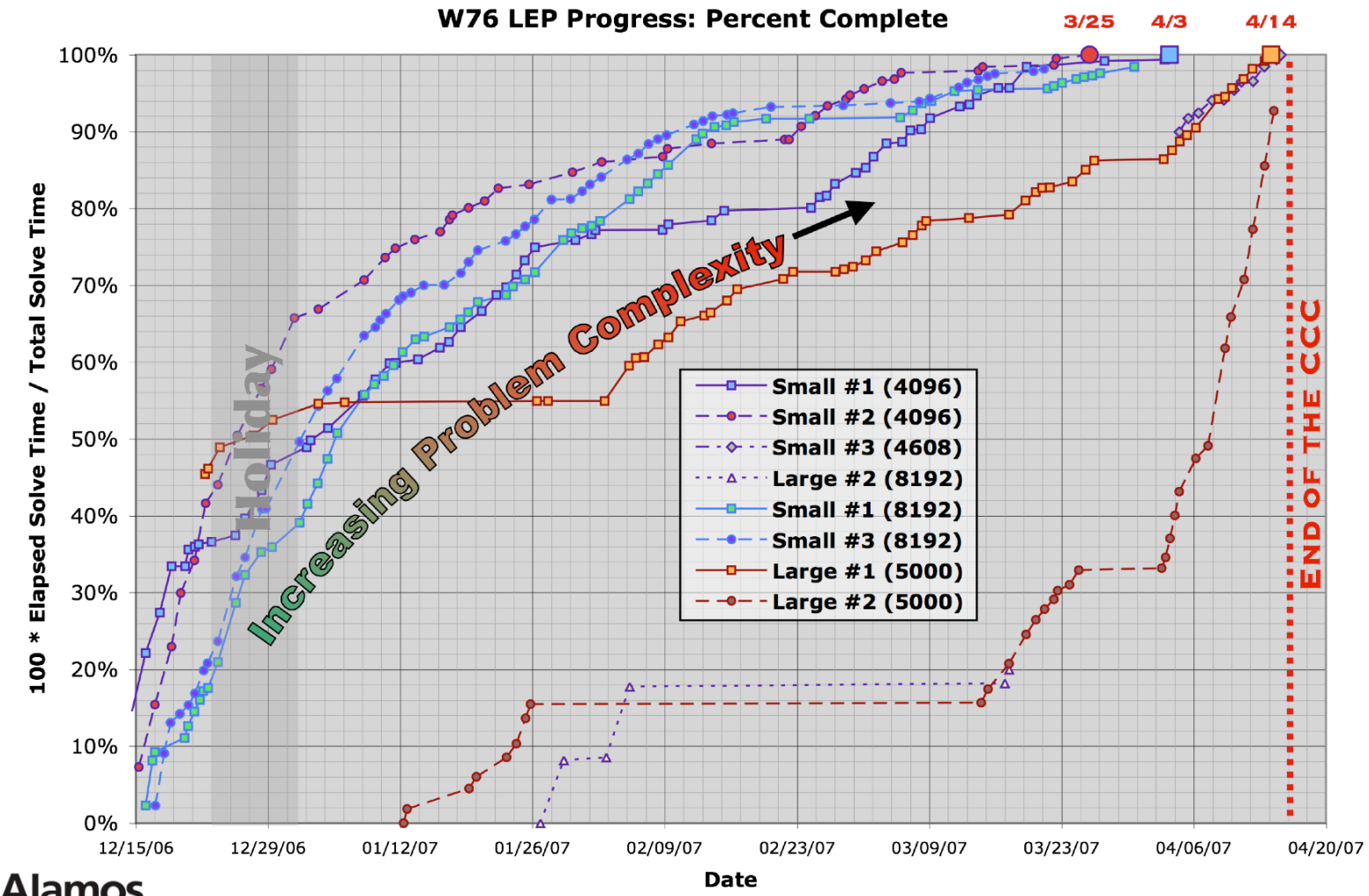
- Motivation: Data Gathered by an "Early Adopter"
- Characterizing the Throughput of an Application
  - Performance and Scaling
  - Efficiency and Utilization
- Quantify the Impact of Reliability on Throughput
  - Optimum Checkpointing Interval
  - Runtime Efficiency
- Strategies to Improve Application Throughput
  - Maximize Ratio of MTTI to Dump Time
  - Minimize Restart Overhead
- **Results Using Application Monitoring**

Automated job monitoring and restart* was successful in keeping user effort on W76 LEP to under 0.25 FTE

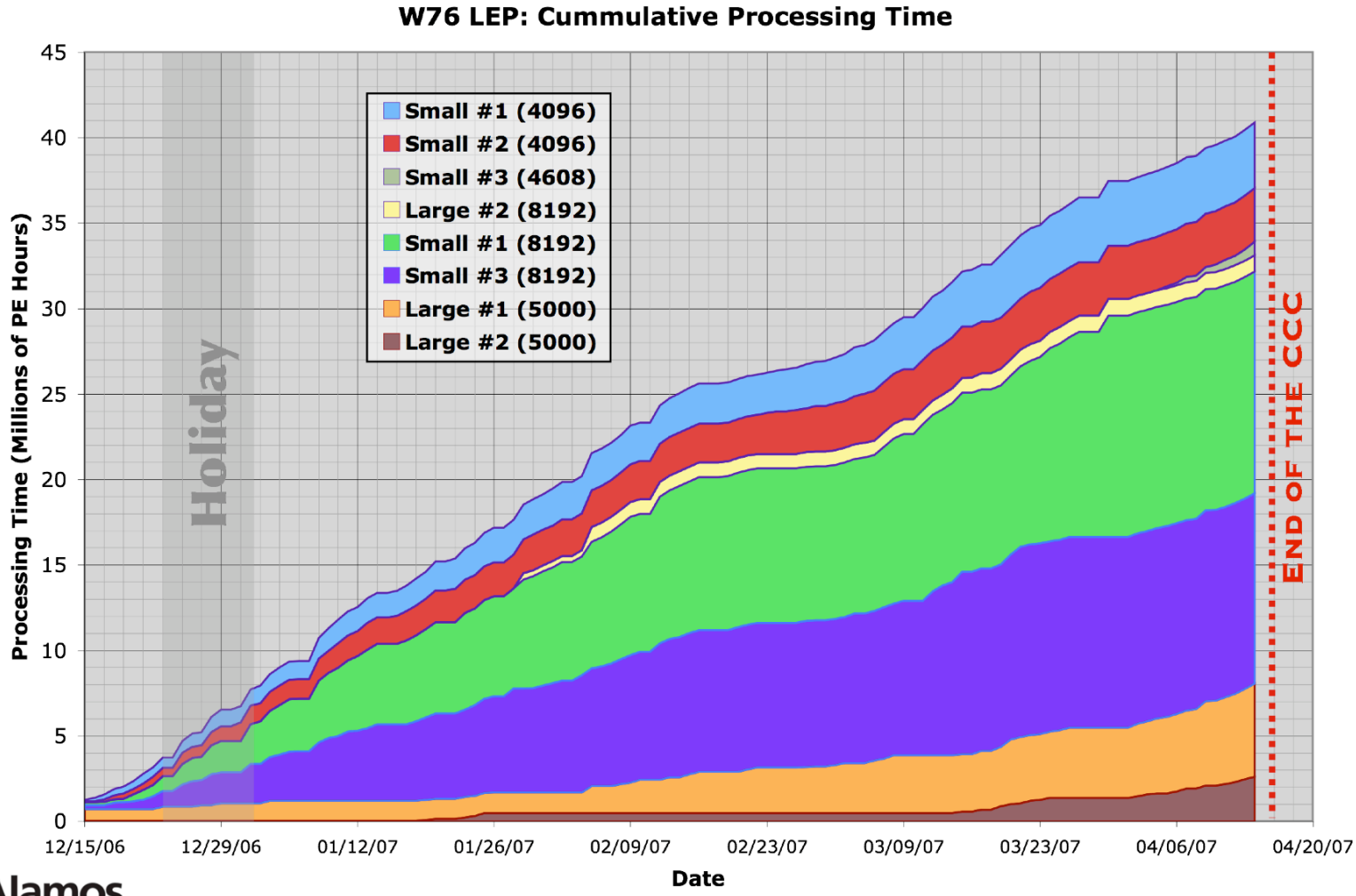
- **Running 41 million PE hours over 16 weeks on Red Storm, Purple, and BG/L, it was typical to restart applications 10-20 times per day**
  - System hardware (nodes, disk controllers, interconnect)
  - System software (job scheduler, queue limits, file system)
  - Application errors
  - Human error
- **The applications are kept running by an automated job monitoring and restart script**
  - Tracks progress by monitoring output directories and killing hung jobs
  - Resubmits interrupted jobs but bails out for recurring failures
  - Uses cron to restart itself even after a full system reboot
- **Job monitoring provided a cheap, portable, and low-overhead means for gathering data to measure AMTTFE and system utilization**

* J. T. Daly, "Facilitating High-Throughput ASC Calculations", *ADTSC Nuclear Weapons Highlights '07*

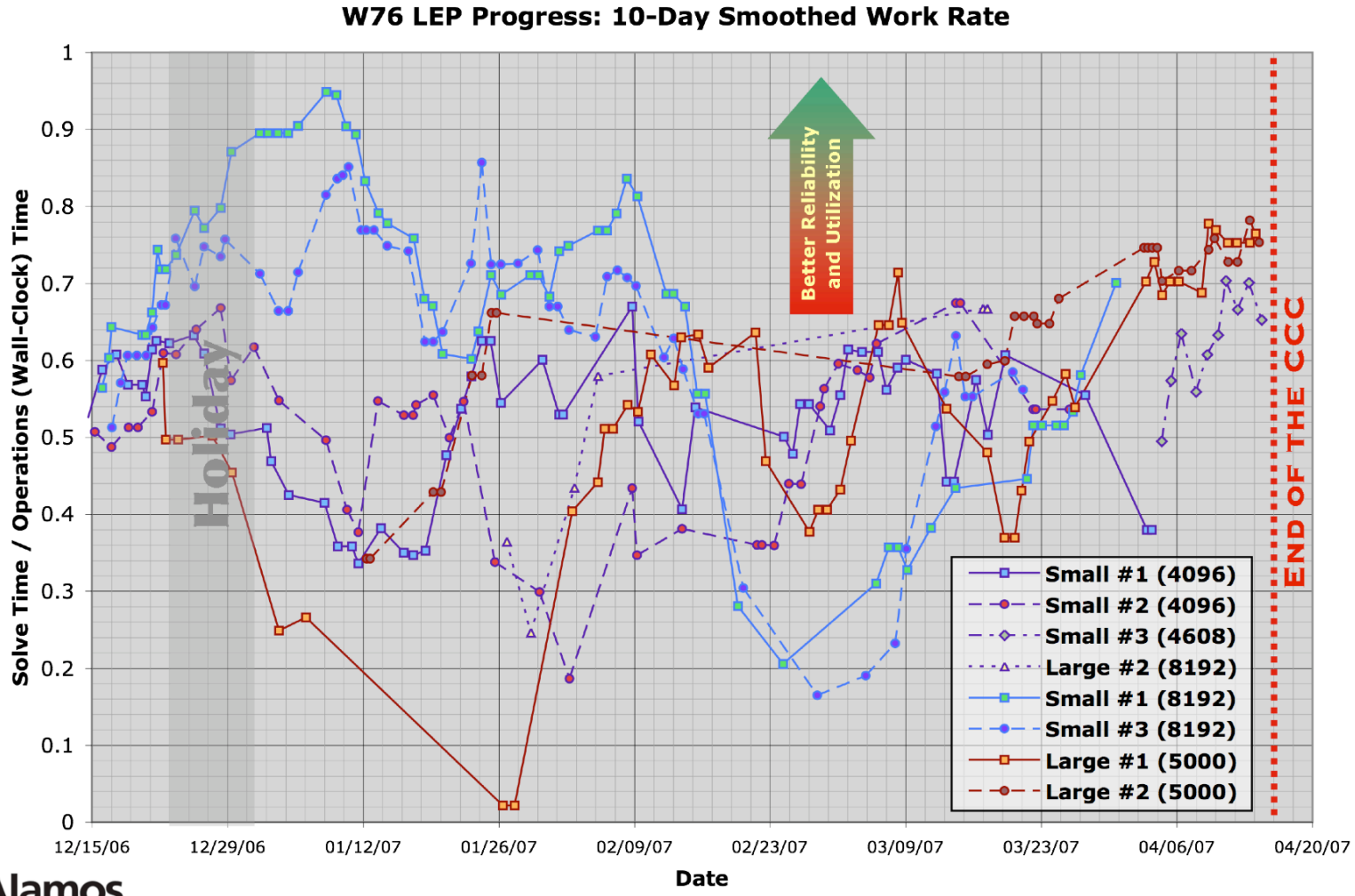
Job Suite was used to manage, monitor, and restart multiple jobs on multiple platforms



Job Suite facilitates steady computational progress at predictable rates by automating application throughput



Job Suite allows applications to realize high average runtime efficiencies in spite of frequent interrupts



Job Suite insulates applications from volatile reliability and availability encountered on truculent systems